Trabalho Prático 1 - Redes de Computadores

Daniel Souza de Campos – 2018054664

UFMG – 2021/2

Sumário

1 – Descrição do Trabalho

O trabalho consiste em desenvolver uma aplicação Cliente-Servidor usando soquetes TCP/IP em *C* com bibliotecas para sistemas **nix*. O sistema simula interações entre um treinador Pokemon, o Cliente, com a sua Pokedex, o Servidor. A Pokedex representa um lugar onde o treinador Pokemon pode guardar informações sobre os Pokemons de interesse. No caso do trabalho, a única informação que a Pokedex irá guardar sobre os Pokemons será o seu nome.

Para se comunicar, o Cliente deve enviar comandos bem definidos para o Servidor e este, por sua vez, deve avaliar a corretude dos comandos, executá-los, alterando ou não a Pokedex, e enviar uma mensagem apropriada de volta para o Cliente. Apenas um Cliente por vez poderá se conectar ao Servidor.

2 – Ferramentas utilizadas

O trabalho foi desenvolvido utilizando a linguagem *C* no editor de código *Visual Studio Code* em um sistema *Windows* mas com o *Windows Subsystem for Linux* instalado, o que permitiu o desenvolvimento e execução do código específico para sistemas **nix*.

Para compilação, foi utilizado o *gcc*. Para verificação de vazamentos de memória, foi utilizado o *Valgrind*. Para uso como repositório e controle de versões, foi utilizado o *Github* cujo *link* é https://github.com/Pendulun/PokedexRedes e ficará público após a data final de entrega do trabalho, dia 21/11/2021.

A máquina usada para desenvolvimento é um notebook DELL, 8GB RAM, processador Intel core i5 7° geração.

3 – Compilando e executando

Para compilar o programa e gerar os binários do Cliente e do Servidor, basta executar o comando *make* dentro da pasta do projeto que contém um *Makefile* com os comandos necessários de compilação. Dessa forma, dois binários, *client* e *server* serão gerados.

Para que o sistema funcione, é necessário que se execute a aplicação do servidor primeiramente. Isso pode ser feito via: *./server <v4|v6> <número de porta>*

Exemplo: *./server v4 51511*

Nenhuma mensagem aparecerá na tela, mas isso significa que o servidor está esperando uma conexão de um cliente.

Com o servidor rodando, pode-se executar a aplicação do cliente no seguinte padrão: *./cliente <IPv* address> <número da porta>*. O valor do primeiro argumento depende do valor do primeiro argumento que foi passado para o executável do servidor.

Exemplo: *./cliente 127.0.0.1 51511*

Nenhuma mensagem aparecerá na tela, mas isso significa que o Cliente está esperando uma mensagem a ser digitada para que seja enviada para o Servidor.

4 – Protocolo da aplicação

Nessa seção, os comandos para a comunicação entre Cliente e Servidor são descritos juntamente com decisões de projeto que influenciam na sua execução.

Existem cinco comandos diferentes que o Cliente pode enviar para o servidor. Algumas regras devem ser seguidas:

- Todos os comandos e nomes de Pokemons deve ser *lower-case*.
- Todos os nomes dos Pokemons têm tamanho limite de 10 caracteres e eles não podem conter nenhum símbolo especial como hífen, hashtag ou asterisco.
- Se o cliente enviar algum comando desconhecido pelo servidor, ele será desconectado.
- O conteúdo da Pokedex está ligado somente ao período de atividade do servidor. Dessa forma, todos os possíveis clientes compartilham da mesma Pokedex.
- Todas as mensagens enviadas trocadas no sistema têm tamanho máximo de 500 bytes. Assume-se que esse tamanho nunca será ultrapassado, não tratando casos que ultrapassem esse limite.
- A Pokedex poderá ter, no máximo, quarenta nomes de Pokemons salvos ao mesmo tempo.
- Assume-se que nunca faltará um nome de Pokemon para ser lido nos comandos, não tratando casos fora desse padrão.

4.1 – Adicionar Pokemon

Um Cliente consegue adicionar até 4 Pokemons ao mesmo tempo utilizando o mesmo comando de adicionar Pokemons. **Quaisquer outros nomes que sejam enviados ao mesmo tempo serão ignorados pelo servidor**. O padrão do comando é o seguinte:

add pokemon1 pokemon2 pokemon3 pokemon4

O Servidor realizará a seguinte ordem de verificações em cada um dos nomes passados, o que influenciará na mensagem de resposta de falha, caso aconteça, ao Cliente:

1. Nome inválido: Tamanho maior que 10 ou possui caracteres especiais
2. Se a Pokedex já atingiu o limite de Pokemons salvos
3. Se o Pokemon já está presente na Pokedex

4.2 – Remover Pokemon

Um Cliente consegue remover um Pokemon por vez da Pokedex. **Quaisquer outros nomes enviados ao mesmo tempo serão ignorados pelo servidor.** O padrão do comando é o seguinte:

remove pokemon

O Servidor irá realizar, em ordem, as seguintes verificações no nome do Pokemon, que influenciam em uma possível mensagem de erro:

1. Nome inválido: Tamanho maior que 10 ou possui caracteres especiais
2. Se a Pokedex tem tamanho zero
3. Se o Pokemon está presente na Pokedex

4.3 – Trocar um Pokemon por outro

Um Cliente consegue trocar o nome de um Pokemon salvo por um outro nome de Pokemon. Dessa forma, são esperados dois nomes de Pokemons para esse comando. Da mesma forma que os comandos anteriores, **quaisquer outros nomes enviados serão ignorados pelo servidor**. O padrão do comando é o seguinte:

exchange pokemon1 pokemon2

O *pokemon1* representa o Pokemon já presente na Pokedex e o *pokemon2* representa o novo nome que vai substituí-lo. O Servidor realiza a seguinte ordem de verificações nos nomes dos Pokemons:

1. Se o *pokemon1* ou o *pokemon2* possuem nomes inválidos
2. Se a Pokedex tem tamanho 0
3. Se o *pokemon1* realmente existe na Pokedex
4. Se o *pokemon2* já existe na Pokedex

4.4 – Listar os Pokemons salvos

Um Cliente pode pedir uma lista de nomes ao Servidor relativos aos nomes dos Pokemons presentes na Pokedex em qualquer momento. O comando é o seguinte:

list

O Servidor testa apenas se a Pokedex está vazia e, se sim, a mensagem *none* será enviada de volta ao Cliente.

4.5 – Matar o servidor

O Cliente pode se desconectar a qualquer momento do servidor, ao fechar o seu próprio terminal ou encerrar a sua execução, que o Servidor continuará funcionando e aceitando novas conexões. Se o Servidor for desligado, ao fechar o seu terminal ou encerrar a sua conexão, o cliente irá se desconectar também.

O seguinte comando existe para que o Cliente informe que o Servidor deve ser desligado e desconectar todos os Clientes:

kill

Ao receber esse comando de um Cliente, o Servidor irá desconectar o Cliente e se desligar, perdendo as informações sobre a Pokedex.

5 – Sobre o código de implementação

A base do código foram as aulas de programação de Sockets TCP/IP em C que o professor disponibilizou no Youtube. O código dessas aulas implementava um servidor de echo que já tratava a questão de aceitar endereços *IPv6* ou *IPv4* pelo servidor, iniciar o Servidor e o

Cliente e trocar apenas uma mensagem entre os dois. Dessa forma, foi necessário adaptar o código original para que:

- O Servidor aceite várias mensagens do Cliente
- O Cliente envie várias mensagens ao Servidor
- Caso um Cliente se desconecte, o Servidor fique esperando por novas conexões
- O Servidor trate dos comandos recebidos, gerencie a Pokedex e responda apropriadamente o Cliente

Além das aulas em vídeo, também foi referenciado o livro *TCP/IP Sockets in C, Practical Guide for Programmers 2 Ed* de Michael Donahoo e Kenneth Calvert.

6 – Principais dificuldades

De acordo com a necessidade de se adaptar o código das aulas de programação de soquetes TCP/IP em C, foram surgindo dificuldades. A primeira dificuldade foi a de permitir que o Servidor e o Cliente pudessem trocar várias mensagens entre si tratando o caso de receber uma mensagem em vários pacotes. Depois disso, foi necessário fazer com que, ao detectar que o Cliente se desconectou, o Servidor esperasse por outra conexão.

Passando para a parte do conteúdo das mensagens em si, também tive dificuldades para realizar a *tokenização* e detectar qual era a semântica da mensagem. Criar a Pokedex em formato de lista duplamente encadeada e implementar os métodos de sua manipulação não foram um problema muito grande.

A programação em C, a qual eu não praticava há algum tempo, com certeza foi a maior dificuldade. A maior parte do tempo investida no desenvolvimento do trabalho foi devido a debuggar código e tentar entender porque as coisas não funcionavam ou porque funcionavam.

6 – Conclusão

Esse trabalho tinha o objetivo de implementar um sistema em rede de Cliente-Servidor que usasse soquetes TCP/IP em C e que a sua comunicação fosse realizada por meio de mensagens padronizadas caracterizando o protocolo da aplicação. Acredito que esse objetivo tenha sido alcançado visto que o sistema desenvolvido passou em todos os testes disponibilizados.

No geral, a implementação do programa levou mais tempo do que o esperado tendo a dificuldade em programação em C como principal causadora desse mal. Sobre o código, tentei seguir boas práticas gerais de programação.

Uma análise da implementação é que a verificação do padrão dos comandos é feita totalmente no lado do Servidor. Acredito que isso também poderia ser feito no lado do Cliente ou até mesmo parcialmente em ambos os lados com, por exemplo, o Cliente verificando o comando em si e o Servidor verificando os nomes dos Pokemons e alterando o estado da Pokedex. Dessa forma, caso o servidor aceitasse vários Clientes ao mesmo tempo, uma parte da carga das verificações seria transferida para os Clientes contribuindo para a performance do Servidor como um todo. Entretanto, não sei o quão seguro seria fazer isso por desconhecer boas práticas de segurança em redes programadas em C além de não usar a função *printf* para imprimir mensagens na tela advindas do cliente, mas sim, usar a função *puts*.